Proceedings of the 2006 IEEE/RSJ
International Conference on Intelligent Robots and Systems
October 9 - 15, 2006, Beijing, China

Mission control system for dam inspection with an AUV

Narcis Palomeras, Marc Carreras, Pere Ridao, Emili Hernandez
University of Girona
Edifici Politecnica IV, Campus Montilivi
17071 Girona, Spain
Email: npalomer@eia.udg.es

***Abstract*— This paper presents a complete control architecture that has been designed to fulfill predefined missions with an Autonomous Underwater Vehicle (AUV). The control architecture has three levels of control: mission level, task level and vehicle level. The novelty of the work resides in the mission level, which is built with a Petri network that defines the sequence of tasks that will be executed depending on the unpredictable situations that may occur. The task control system is composed of a set of active behaviours and a coordinator that selects the most appropriate vehicle action at each moment. The paper focuses on the design of the mission controller and its interaction with the task controller. Simulations, inspired on an industrial underwater inspection of a dam grate, show the effectiveness of the control architecture.**

I. Introduction

A mission controller is the part of a control architecture that is in charge of defining high-level phases to be carried out in order to fulfil a predefined mission. Autonomous Underwater Vehicles (AUVs) are usually developed to observe underwater environments. Due to high costs and severe difficulties in these kinds of autonomous vehicles, mission controllers must strongly focus on the security of the vehicle, as well as, on its performance to fulfil the mission. Several mission control systems for AUV have been designed over the past decade. In 1994, the ISR, from Portugal, started the development of a management system for the AUV MARIUS that contained a mission management system [1]. The system was based on Petri nets, and was in charge of activating vehicle primitives, needed to carry out the mission. Simultaneously, the NPS from Monterrey was developing a hybrid control system composed by three layers, using the Prolog language as a rule-based mission specification in the higher layer [2]. Another control architecture, which has a mission control system called Helm, is the MOOS architecture designed by Paul M. Newman [3]. Helm decides the most suitable action commands from a set of prioritized mission goals and the current state of the navigation process. Other mission controls systems that have been proposed in the literature are the "SAUVIM task description language" by the ASL from Hawaii [4], the ORCA architecture by Roy M. Turner [5] and the architecture developed by ISE from Canada [6].

This paper describes a complete control architecture that has been designed to fulfil predefined missions with an AUV. The control architecture has three levels of control: mission level, task level and vehicle level. The mission controller is built with a Petri Net which defines the sequence of tasks. Each node of the Petri Net represents a behaviour that is executed on the task controller. Each behaviour has a simple goal such as: move to point, keep depth, avoid obstacle Active behaviours generate a control action on one or more degrees of freedom. According to the active behaviours and some priorities, the task controller generates a coordinated control action that is sent to the vehicle controller. Finally, this low-level controller implements a classical velocity control loop for each degree of freedom.

The novelty of this work resides on the design, implementation and experimentation of the mission control system. The rest of the control architecture has already been tested and presented in previous works [7]. The main distinctive feature of the presented approach, compared with other AUV mission controllers, is its connection and relation with the task level controller. The mission controller does not determine the actions that will guide the robot, but the active behaviours and configuration which, through the task controller, will be coordinated to guide the robot. This common structure in artificial intelligence or in mobile robots, is not usually applied in AUVs, where more classical control theories are applied. The missions for which classical AUV mission controllers are usually developed have unstructured and very large environments to explore. In such situations, the AUV trajectory is calculated to optimize the energy efficiency. The proposed mission controller should have interesting advantages in missions where the robot reactivity to the environment is more important, rather than the energy efficiency. For example, an autonomous inspection of an structured environment, such as a dam, in which the robot must avoid the walls while following a trajectory.

This paper presents the design, simulations and preliminary real results of a mission control system for dam inspection using an AUV. The goal of the project is to develop a complete control architecture that is able to autonomously explore the wall of a dam, see Figure 1, and to build a visual mosaic [8] of it. Previous work on dam inspection has already showed interesting applications using a Remotely Operated Vehicle (ROV) [9]. The use of an AUV having a high accuracy positioning system could entail a fast procedure, a complete scanning of the wall and the georeferencing of the images. The work presented in this paper is centered on the mission control

1-4244-0259-X/06/$20.00 ©2006 IEEE

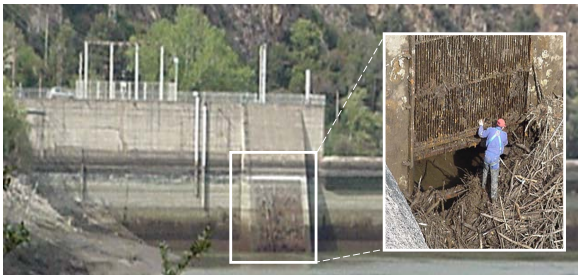


Fig. 1. Wall of an empty dam during maintenance works.

system. Simulations show the achievement of a predefined mission with the Neptune simulator [10]. Preliminary real results using the GARBI AUV show the control of the robot in a pool environment.

The paper is structured as follows. After the introduction section, a detailed description of our control architecture and mission control system will be given in section 2. Section 3 will summarize the most relevant details about the GARBI AUV, the software architecture and the Neptune simulator. The simulated mission and the obtained results will be presented in section 4. Finally, section 5 will conclude the paper.

II. Control Architecture

The finality of a control architecture is to move the robot autonomously to fulfill a set of goals in a particular order and with some constraints. The result of the whole process at every moment is the movement of the robot in each degree of freedom. Figure 2 shows the schema of the three levels that form the proposed control architecture.

A. Vehicle level: Velocity controller

Since the final task is the movement of the robot, the control architecture has, at its lower level, a classic velocity controller. This controller receives the velocity set points from the task level controller and the measured or estimated velocities from a navigation module. The object in charge of doing this task is the PID velocity controller. This object reads the vehicle velocity from the Navigator object (see section 3.1) and receives the velocities set points from the Coordinator Object (see section 2.2). The control system is based on a classical PID controller implemented as a matrix, which also contains the propeller distribution and coefficients [11]. This general implementation allows an easy adaptation to any AUV model. The propeller velocities are finally calculated, converted to voltages and sent to the corresponding electronics.

B. Task level: Behaviours + Coordinator

A common methodology to implement a control architectures is to build a library containing all basic functions that the robot can make. By joining these basic functions it is possible to carry out complex tasks. These basic functions have been named: vehicle primitives, tasks, command primitives... One of the most popular names to express this concept is behaviour, which was introduced in 1986 by Brooks [12] and has been

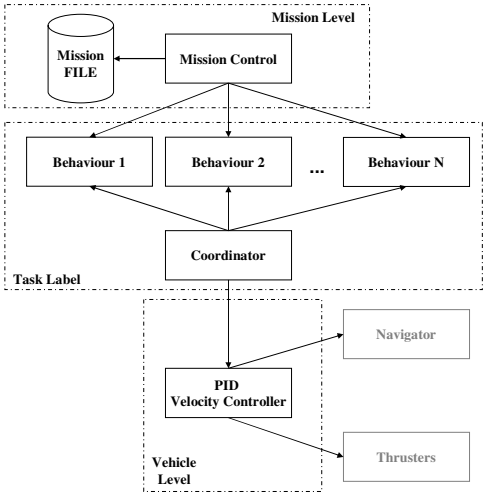


Fig. 2. Proposed Control Architecture.

extensively used. The task level controller presented in this paper is based on very simple behaviours. Since more than one behaviour can be activated, a coordinator is always needed in all no pre-emptive architectures. The coordinator is the object in charge of combining the outputs of all active behaviours to generate a single output.

Each behaviour is an autonomous process with a particular goal. The input of a behaviour can be taken from any object of the software architecture (sensors, actuators). The output contains:

- *Velocity for every DoF:* Shows the desired velocity in every DoF and it is normalized between -1 and 1.
- *Activation level:* Shows the activation level for every DoF. This value is normalized between 0 and 1.
- *Priority:* Every behaviour has a priority level. 0 is the highest priority and there is an inverse proportionality between the priority and the value. The activation level together with the priority is used by the coordinator to combine the outputs of all behaviours.
- *Block:* This is a Boolean value that shows if the behaviour is blocking the execution thread of the mission controller.

To initialize a behaviour, besides particular parameters, it is needed to setup the following values:

- Enable: Boolean variable that indicates if the behaviour is activated or not and, therefore, if its output will be considered by the Coordinator.
- Priority: Priority that will have the output of this behaviour.
- TimeOut: The time out indicates when the behaviour will block the execution thread. If TimeOut<0, the behaviour blocks the execution thread until its goal is fulfilled. If TimeOut=0, the behaviour doesn't block the execution thread. If TimeOut>0, the behaviour blocks the execution thread until TimeOut seconds or until its goal is fulfilled.

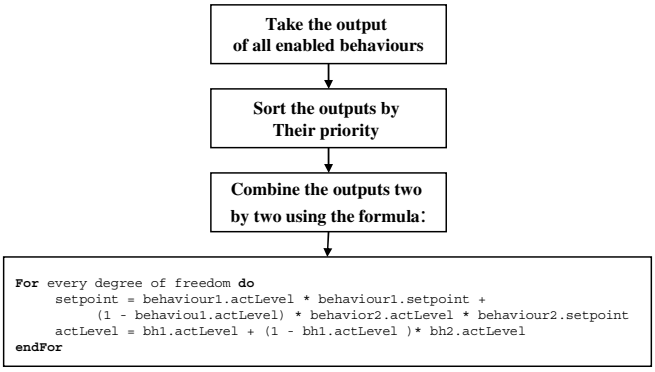


Fig. 3. Coordinator procedure.

The object Coordinator is in charge of taking all the enabled behaviour outputs to combine them into a single one, which will be sent to the velocity controller. To combine all the outputs, the coordinator follows the schema showed in Figure 3. Using this coordinator, if the activation value of all active behaviours is 1 (max. value), the coordinator output corresponds to the behaviour output with more priority (preemptive architecture). On the other side, if the activation values are less than 1, the final output will be the combination of all active behaviours (collaborative architecture). Since each DoF is treated separately it is possible, by using activations levels with value 0, to program behaviours that do not affect all DoFs. The coordinator output, after combining all active behaviours, is a vector as large as the number of DoFs of the robot. Each value corresponds to a normalized velocity.

C. Mission level: Mission Control System

The task controller decides how to guide the robot movements in each situation. However, to carry out medium-high complex missions it is very difficult to design a unique set of behaviours that can accomplish it. In these missions, it is necessary to have an autonomous system able to enable/disable and reconfigure behaviours. Our mission controller uses a Petri Net to accomplish the mission plan. A Petri Net has place nodes, transition nodes, and directed arcs that connect places with transitions. It is represented as a graph defined by a quadruple, see Equation 1:

$$PN = (P, T, I, O) \tag{1}$$

where P is a finite set of places, T is a finite set of transitions, $I(p,t)$ is a mapping corresponding to the set of directed arcs from places to transitions, and $O(t,p)$ is a mapping corresponding to the set of directed arcs from transitions to places.

We use a kind of Petri Net called *marked graphs* which is a pure ordinary Petri net system where every place has only one input transition and one output transition, see Equation 2:

$$\forall p \in P : |\bullet p| = |p\bullet| = 1 \tag{2}$$

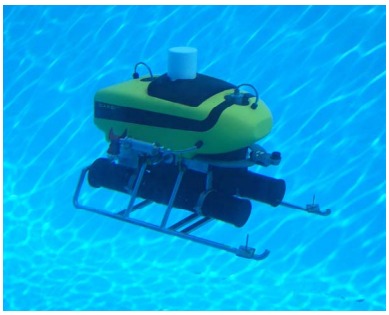


Fig. 4. GARBI^{AUV} in the laboratory facilities.

where $|\bullet p|$ is the number of inputs of place p and $|p\bullet|$ is the number of outputs of place p.

In our Petri net, every place corresponds to one behaviour with a particular configuration. When a place has a token, this behaviour is enabled. When all places that go towards a transition are enabled, and their behaviours do not block the execution thread, the transition is ready to be fired. When a transition is fired, a token is removed from each of the input places of the transition and a token is generated in each output places of the same transition, see Equation 3:

$$\forall pP : M'(p) = M(p) + O(t,p) - I(p,t) \tag{3}$$

where M is a n-dimensional integer vector which assigns a non-negative integer number of tokens to each place of the net.

A Petri net can be represented as a matrix, see Equation 4, called the incidence matrix (C). In addition, if active transitions and the actual state are known, it is possible to calculate the new state, see Equation 5.

$$C_{ij} = O(t_j, p_i) - I(p_i, t_j) \tag{4}$$

where $1 < i < (sizeofP)$ and $1 < j < (sizeofT)$.

$$M_{i+1} = M_i + CT \tag{5}$$

Therefore, the control mission algorithm starts on the initial state M_i, checks fired transitions, applies Equation 5, and repeat this process until the final state M_f is reached.

At this moment, the generation of a petri Net and it's verification is a manually process carried out by the user. In a medium term, the system will be redesigned to automatically plan the missions according to some a priori knowledge.

III. GARBI AUV, SOFTWARE ARCHITECTURE AND NEPTUNE SIMULATOR

The work presented in this paper has been designed to be fulfilled with the Autonomous Underwater Vehicle GARBI^{AUV}. Simulations have been performed using an accurate model of the robot [13]. GARBI is an underwater robot equipped with several batteries, 5 propellers, two computers, a DVL navigation sensor, an imaging sonar, a video camera and a DGPS sensor for global positioning when navigating

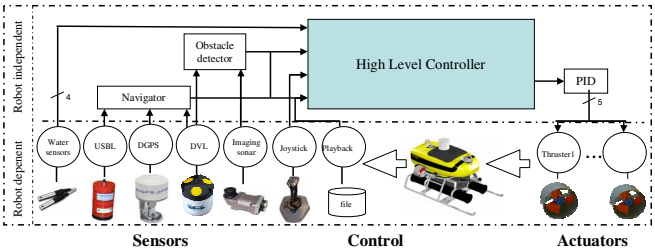
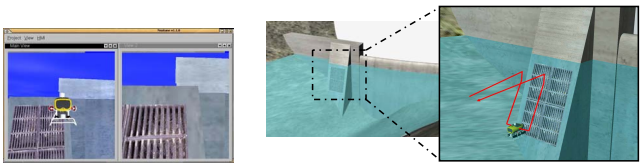


Fig. 5. GARBI^{AUV} Software architecture



a) b)

Fig. 6. a) Neptune screen shot during the simulation. b) Predefined mission schema.

at surface (see Figure 4). Also, GARBI^{AUV} has an external USBL sensor that is used to localize it with high precision, and to communicate with it using an acoustic modem. This section describes the software architecture and the used simulator.

A. Software architecture

The software architecture has the task of guaranteeing the AUV functionality. It is build with a set of CORBA-RT objects. These objects are executed in the two robot PCs and an external PC. The architecture is composed by a base system and a set of objects (see Figure 5). The goal of the base system is to control all processes and to register data. The base system is composed by the base objects, which are periodic threads and logger systems. The threads are inherited by the final objects that must be executed periodically (for example the DVL sensor). The logger system is used to acquire data from sensors, actuators or any other object.

Most of the objects implement sensors or actuators. There is one object for each propeller. There is also one object for each real sensors (see Figure 5). Moreover, navigation, perception or control systems are also implemented as independent objects. The navigator object has the goal of estimating the position and attitude of the robot from the DVL and DGPS sensors. The obstacle detector object has the goal of detecting horizontal obstacles with the imaging sonar, and the sea bottom obstacle with the DVL sensor. The control architecture is implemented as different independent objects. The vehicle controller is contained in the PID object. The task and mission controllers are contained in the high-level controller objects.

The software architecture provides a methodology to communicate all the systems and to execute them at different periodic threads.

B. Neptune simulator

Neptune (see Figure 6a) is a simulation software for underwater robots. It allows testing the software developed for an AUV without using the real robot. Neptune communicates with the software architecture as a client-server application. The server is an object called UUVModel which simulates the dynamics model of an Unmanned Underwater Vehicle (UUV). This object only requires the parameters of the dynamics, which are entered with a configuration file [10]. This object returns the position, velocity and acceleration of the robot, which is used by the client for graphical representation. UUVModel also allows the simulation of several sensors.

C. Used behaviours

The mission consists in inspecting a grate mounted on the wall of a dam using the GARBI^{AUV} and the simulation software Neptune.

D. Mission description

To verify the state of the grate, the AUV will move to a initial known position, which is located at surface level over the grate (see Figure 6b). Then it will submerge until the highest point of the grate, which is also a known position. In this movement the AUV will maintain a security distance from the wall and it will constantly face it to acquire the images properly. At this point, the robot will start the inspection movement which consists in a U movement avoiding the bottom and maintaining the security distance from the wall. Note that the wall of the dam has an inclination with respect to the vertical.

set of behaviours. Each of them is in charge of accomplishing a simple goal. The control architecture can work in two modes: REAL and VIRTUAL. Since the mission was executed in simulation, the architecture was executed in VIRTUAL mode and the Neptune simulator was required. However, the implementation of the behaviors does not depend on the execution mode. This means that the same implemented behaviours could be used in real experiments. The execution mode affects the Navigator object, which in VIRTUAL mode takes the data from Neptune objects, and in REAL mode takes the data from real sensors.

The implemented behaviour are:

- *KeepDepth:* Keep a constant depth. Navigator takes the current depth from the pressure sensor in REAL mode or from the simulator in VIRTUAL mode.
  - Parameters(depth, velocity, priority, timeOut)
- *MoveTo2D:* Move the vehicle to a specific 2D point. In REAL mode the position is taken from the DGPS in surface and from the USBL when the vehicle is underwater and in VIRTUAL mode the Navigator object takes the position from the simulator.
  - Parameters(X, Y, priority, timeOut)
- *KeepAltitude:* Maintains the vehicle at a certain distance from the bottom. Navigator takes the current altitude from the DVL in REAL mode, and from the simulator in VIRTUAL mode.
  - Parameters(altitude, priority, timeOut)

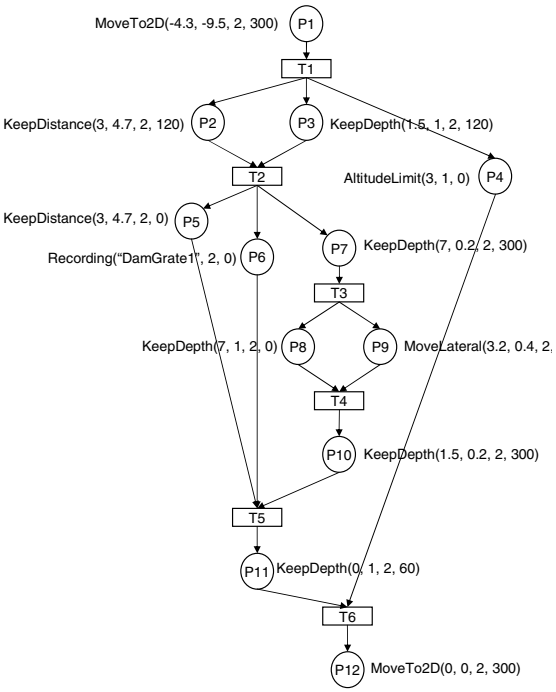


Fig. 7. Petri Net used to fulfill the mission

- *KeepDistance:* Maintains the vehicle at a certain tation and at a certain distance from the nearest found in that orientation. This behaviour takes the tation from the Navigator object, that reads the orie from the magnetic compass in REAL mode or fr simulator in VIRTUAL mode, and the distance rrom the imaging sonar sensor in REAL mode or from the simulator in VIRTUAL mode.
  - Parameters(distance, orientation, priority, timeOut)
- *MoveLateral:* Move the vehicle transversally a certain distance maintaining the current orientation. Takes the position from the Navigator like the MoveTo2D behaviour.
  - Parameterers(distance, velocity, priority, timeOut)
- *Recording:* Enables the camera. The images taken by the camera are saved in the hard drive. This is an special behaviour because it does not affect the vehicle motion.
  -Parameters(folderName, priority, timeOut).

E. Mission controller

To implement the mission controller it is required to define several aspects:

- Petri Net with the sequence of behaviours.
- Parameters of every place/behaviour.
- Initial and final states in the Petri Net.

The Petri Net and the specific behaviours and parameters of each state are showed in Figure 7. Equation 6 shows the incidence matrix of the Petri Net (see Epuation 4).

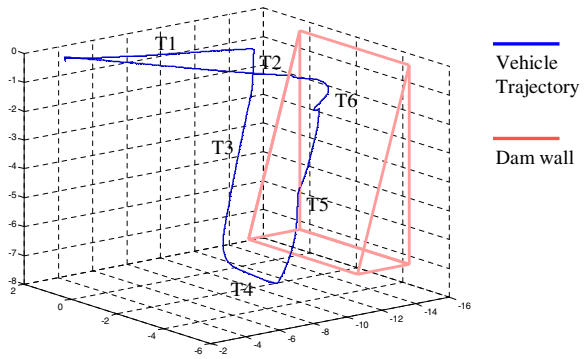


Fig. 8. Trajectory of the AUV during the simulation.

$$C_{i,j} = \begin{bmatrix} -1 & 0 & 0 & 0 & 0 & 0 \\ 1 & -1 & 0 & 0 & 0 & 0 \\ 1 & -1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & -1 \\ 0 & 1 & 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & 0 & -1 & 0 \\ 0 & 1 & -1 & 0 & 0 & 0 \\ 0 & 0 & 1 & -1 & 0 & 0 \\ 0 & 0 & 1 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 & -1 & 0 \\ 0 & 0 & 0 & 0 & 1 & -1 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \quad (6)$$

The initial and final states are:

$$M_i = (1,0,0,0,0,0,0,0,0,0,0,0,0)$$
$$M_f = (0,0,0,0,0,0,0,0,0,0,0,0,1)$$

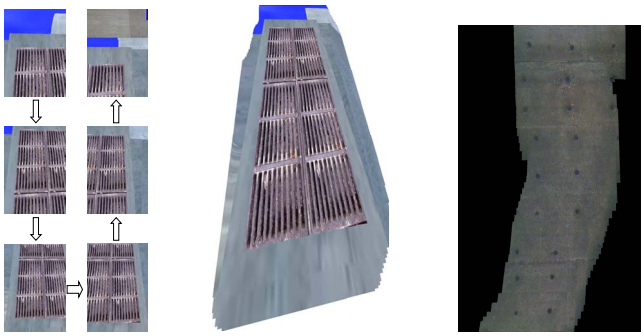
F. Results

In order to simulate the dam inspection mission, the six behaviours and the mission controller were implemented and integrated in the software architecture. In this preliminary work, the mission was simulated using Neptune. Nevertheless, some behaviours were tested in reality with the GARBI^{AUV}.

In the simulation, the dynamics parameters of GARBI were used [13]. The simulation was executed online and it took 6 minutes to accomplish the whole mission. Figures 8 and 6a show the obtained trajectory and a screenshot from Neptune during the mission. It must be noted that the designed mission control system was able to guide the AUV to accomplish the mission. Figure 8 shows the evolution of active transitions in the Petri Net.

Figure 9 shows some captured images during the simulation and the final mosaic generated off-line. Figure 9c shows a similar result obtained in a previous work on dam inction using the URIS ROV [9]. Similarity between real and simulated mosaics shows the reliability of Neptune software.

Finally, some part of the control architecture has already been tested with the real robot. Figure 10 shows an experiment with the KeepDepth and AltitudeLimit behaviours. It can be observed that the KeepDepth behaviour was controlling GARBI^{AUV} until the AltitudeLimit behaviour became active. In this situation, since the AltitudeLimit behaviour had more priority, the response of the KeepDepth behaviour was subsumed.



a) b) c)

Fig. 9. a) Captured images during the simulation. b) Generated mosaic fron captured images. c) Real dam wall mosaic using URIS ^{AUV}.

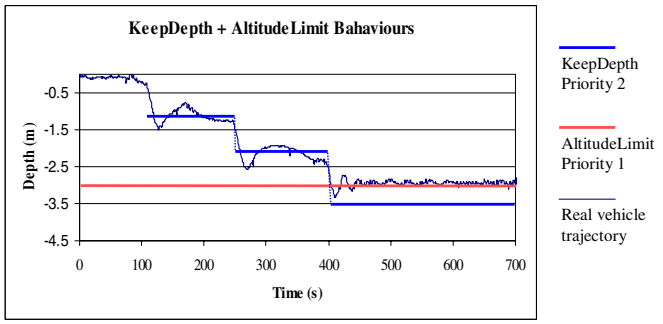


Fig. 10. Real robot trajectory controlled by two enabled behaviours.

IV. Conclusion

This paper has presented a control architecture to carry ou predefined missions with an underwater robot. The work ha been focused in the design of a mission controller that interact with a behaviour-based task controller. The mission controlle is easily built with a Petri Net, which determines activ behaviours that execute each task of the mission. The paper ha detailed implementation aspects and showed simulations on predefined AUV mission, inspired by a dam inspection appli cation. Also, some preliminary real results with two enabled behaviours have been presented. The obtained results prelude good effectiveness and robustness of the mission controller in real experiments. Short term future work consists in preparing the robot for real experimentation and in a medium term future work, the mission control system will be redesigned to automatically plan the missions according to some a priori knowledge, as other mission control systems do.

Acknowledgments

This research was sponsored by the Spanish government (DPI2005-09001-C03-01). The authors would like to thank the company Endesa Generacion for its collaboration with the project.

References

[1] P. Oloveira, A. Pascoal, V. Silva, and C. Silvestre, "Design, development, and testing at sea of the mission control system for the marius autonomous underwater vehicle," in *Oceans MTS/IEEE*, 1996.

[2] D. Marco, A. Healey, and R. Mcghee, "Autonomous underwater vehicles: Hybrid control of mission and motion," *Autonomous Robots*, vol. 3, pp. 169–186, 1996.

[3] P. M. Newman, *MOOS - Mission Orientated Operating Suite*, Department of Engineering Science Oxford University, 2005.

[4] T. W. Kim and J. Yuh, "Task description language for underwater robots," in *Intl. Conference on Intelligent Robots and Systems*, 2003.

[5] R. M. Turner, "Orca: Intelligent adaptive reasoning for autonomous underwater vehicle control," in *Proceedings of the FLAIRS–95 International Workshop on Intelligent Adaptive Systems*, Melbourne, Florida, 1995, pp. 52–62.

[6] M. Kao, G. Weitzel, and X. Zheng, "A simple approach to planning and executing complex auv missions," 1992.

[7] M. Carreras, P. Ridao, R. Garcia, and J. Batlle., *Behaviour Control of UUVs*. G. Roberts and R.Sutton, 2005.

[8] R. Garcia, J. Batlle, X. Cufi, and J. Amat, "Positioning an underwater vehicle through image mosaicking," in *IEEE International Conference on Robotics and Automation (ICRA)*, Seoul, Rep.of Korea, 2001, pp. 2779–2784.

[9] J. Batlle, T. Nicosevici, R. Garcia, and M. Carreras, "Rov-aided dam inspection: Practical results," in *6th IFAC Conference on Manoeuvring and Control of Marine Crafts*, Girona, Spain, September 2003.

[10] P. Ridao, E. Batlle, D. Ribas, and M. Carreras, "Neptune: A hil simulator for multiple uuvs," in *Oceans04 MTS/IEEE*, Kobe, Japan, November 9-12 2004.

[11] T. I. Fossen, *Marine control systems*. Marine cybernetics, 2002.

[12] R. A. Brooks, "A robust layered control system for a mobile robot," *IEEE J. Robot. and Auto.*, vol. 2, no. 3, pp. 14–23, 1986.

[13] P. Ridao, J. Batlle, and M. Carreras, "Model identification of a low-speed uuv with on-board sensors," in *IFAC conference CAMS2001, Control Applications in Marine Systems*, Glasgow, Scotland, U.K, 18-20 July 2001.